JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2009**

Vol. 8, No. 6, September-October 2009

Ecosystems

John D. McGregor, Clemson University and Luminary Software LLC, U.S.A.

Abstract

Software development of almost any industrial strength product involves the collaboration of numerous people, alliances among companies, and dependencies among pieces of code. A successful product spawns other opportunities such as books, videos, and consulting. These interactions form an ecosystem in which changes to one entity can affect many others. In this issue of Strategic Software Engineering I want to explore how we can manage relationships in software development.

1 INTRODUCTION

One of my favorite places on earth is the Blue Ridge Parkway, a unit of the United States national park system. The parkway runs from Virginia's Shenandoah National Park down to the Great Smoky Mountain National Park. It is a 400 plus mile highway that runs through the Blue Ridge mountains. No commercial vehicles are allowed on the road and it has a 45 mile per hour top speed. It is peaceful and there are many places where all you can see is miles of forest.

Increasingly though what you can see from the parkway is the encroachment of industry and housing into the forest. Except for signs on trees and a few boundary markers you can't tell where the parkway ends and the surrounding country side begins. Events, like acid rain, that affect the surrounding areas affect the parkway. Streams that originate in the high altitudes of the parkway flow into civilization seamlessly. Around cities there is more traffic on the parkway as commuters use it as a shortcut to avoid traffic jams.

An ecosystem is a set of organisms and their interactions with the physical world. An ecosystem is often thought of as being complete in the sense that it includes everything for the organisms to survive and perhaps thrive. This can be a rather large number of organisms and an even larger number of relationships among the organisms. The parkway is an ecosystem in which the availability and quality of resources such as water and air affect the animals, birds, and insects.

When one constituent of the ecosystem weakens the rest of the ecosystem adjusts. Right now a particular insect, the woolly adelgid, is destroying hemlock trees along the parkway. The insect began in the Shenandoah Park and then spread down the parkway.

By May of 2009 it had reached the other end of the parkway. With these trees disappearing, areas that were once in shade most of the time now receive direct sunlight. That may cause certain plants to die but others, which previously could not survive there, will take their place and thrive in the sunshine.

Using this phenomenon as a metaphor for software organizations is not original with me[Messerschmitt 03][Miller 06]. I think it provides a way of thinking about the interactions of organizations, teams, and individuals that is useful. Consider the example of the ecosystem encompassing an operating system[Microsoft 09]. No one would use an operating system for which there are no applications and, conversely, no one would produce an application without understanding the operating systems with which it should be compatible and without understanding the market for applications that use the operating system. This symbiotic relationship extends to many areas. An excellent book about how to program in a new language or paradigm will encourage programmers to try it.

Software development takes place in an ecosystem of organizations that are related in many different ways. Seldom can a development manager make any significant decision without impacting other members of the ecosystem. Development managers need a framework in which they can understand the relationships among the elements of the ecosystem. Some activities such as testing need continuity across team and contractual boundaries and managers must be able to trace these activities across these boundaries. I will describe a technique that will allow managers to get their hands around at least some portion of relevant ecosystems.

I avoid chasing the latest fads and concentrate on what adds value. I believe analyzing the relationships among entities that impact your organization as an "ecosystem" is a useful exercise that will give you a perspective that you are not likely to discover otherwise.

- Understanding the organization's position in the ecosystem is directly relevant to managing the technical risks surrounding the product line. An obvious example is the need to manage interfaces to suppliers who will supply essential components. Their failure to deliver on time and to quality will ripple through the supply chain in the ecosystem.
- Forecasting technical changes and analyzing markets are both more accurately done if only the extent of the ecosystem need be considered. Defining the entities in the ecosystem reveals trends and future directions that are essential inputs into strategic decisions. The amplifying effect of the ecosystem can be accounted for when making these forecasts.
- Analyzing the ecosystem gives the manager the opportunity to gain additional understanding of the domain by providing context for reflection. Examining relationships, whatever the reason, usually results in a deeper understanding of the entities that are related.

In this issue of Strategic Software Engineering I will discuss several aspects of strategic decision making in an ecosystem of development organizations. I will illustrate how using this perspective adds value to the manager's portfolio.

2 AN EXAMPLE

One type of software development ecosystem concerns the entities centered around an open source development community. Open standards activities often serve as the pebble in the oyster that stimulates the growth of an ecosystem;however, I will use a different open source example, my favorite - Eclipse [Eclipse 09], to illustrate several technical and business/managerial issues.

The Eclipse "project" started in 2001 as an open source project with a major contribution of code from IBM. A consortium of software vendors gathered to support the project and in 2004 the Eclipse Foundation was formed and houses numerous projects. Among the tasks of the professional staff of the foundation is ecosystem development. By promoting the Eclipse tools and projects, the staff encourages the development of an ecosystem around the foundation. The people who contribute to Eclipse have the opportunity to make money through writing books, giving tutorials, and many other activities.

My university and many other organizations are members of Eclipse. Membership does not provide special access but it does provide visibility and association with a highly regarded organization. Joining the ecosystem either directly through membership or indirectly through use of the assets connects the organization with others in the ecosystem. Perhaps the biggest advantage of being in the ecosystem is influence on, and knowledge of, future directions that might benefit your organization. This knowledge can lower risk by allowing managers to identify assets that may be useful in developing products or marketing campaigns.

The Eclipse ecosystem includes many groups with varying intensities of association to the Foundation. The Eclipse Foundation has a Board of Directors and Requirements and Architecture boards that oversee new and existing projects. The companies that provide members for these boards are tightly tied to the Foundation and have interactions with many of the projects and indirectly with many other companies. Companies that contribute to one or more projects have a more limited interaction but still do interact with other companies. The external companies that free ride still have interactions by posting feature requests and reporting bugs.

The ecosystem of Eclipse is extensive and continues to expand. A website, www.eclipseplugincentral.com, lists available plug-ins, some of which are commercial and some of which are open source. In addition,

- There continue to be new projects initiated within the Eclipse foundation.
- Other open source projects, such as TopCased [Topcased 09], build on top of the Eclipse platform.

- Other organizations provide services, such as training and consulting. EclipseCon is one of several events within the ecosystem.

The architecture of the Eclipse platform is explicitly extensible, which has allowed many companies to benefit by defining plug-in tools that take advantage of the platform capabilities. Whether the organization contributes to Eclipse or free-rides, they join the community in relying on the reputation of Eclipse to add value to their product. They become dependent on the periodic upgrades made to the platform to enable new capabilities and to fix any defects.

To orchestrate the interactions among members of the ecosystem the basic platform team has established a regular rhythm of major releases. This has allowed other organizations outside the foundation but within the ecosystem to establish their own rhythm. An organization such as Topcased plans an annual release a certain number of weeks after Eclipse is due to make its release. The rhythms of the constituents of the ecosystem must be compatible for maximum health of the ecosystem.

A healthy ecosystem builds momentum for all of the constituents of that ecosystem. The activities of one segment contribute to growth in other segments. Of course, disruptive activities in one segment can be amplified throughout the ecosystem. Growth, even due to success, can be disruptive and eventually can lead to shrinkage or to the ecosystem splitting. When Eclipse makes its annual release there is a period afterward where some plug-ins from external companies may experience problems, even though release candidates are available well in advance of the release date.

One responder to a blog posting recently claimed that “ecosystem implies survival of the fitest” and that contradicts how humans work. I disagree. An ecosystem is a sufficiently complex entity that any action taken to block the demise of a constituent may have unexpected consequences. Some projects in an open source organization thrive and attract committers while others languish from lack of contributions. It is survival of the one most in synch with the rest of the ecosystem.

3 METHOD FOR MODELING INTERACTIONS

To investigate the workings of an ecosystem we need to identify the organizations that are significant to the specific analysis and the interactions among these groups. [Brinkkemper 07] provides one technique termed a Software Supply Network. I will use the concept of a *transaction* defined by Baldwin [Baldwin 07] as the unit of interaction for this analysis. While Baldwin used this technique to examine the boundaries between organizations, I will use the analysis of transactions to study the dependencies among organizations and to consider the impact of existing boundaries between teams in the ecosystem. I believe this technique has a bit more power than Brinkkemper’s approach. In this section I will briefly describe this approach before applying it in the next section.

According to Baldwin, two types of activities are of interest in evaluating the boundaries between organizational units. One is termed a “transaction.” A transaction is a flow across boundaries between groups and incurs costs because the flow crosses a boundary, e.g. the preparation of contracts or additional documentation. The second type of activity is a "transfer," which is a flow within boundaries that incurs only incidental costs, such as the time required for a meeting. A transaction-free zone is where all exchanges are transfers, i.e., have only incidental costs. Baldwin states that transactions happen across the boundaries of organizations while an organization or team is a transaction-free zone.

Software development organizations have many boundaries because they work with many different units. For example, a development team is within a larger development organization. That organization is within a business unit and will at the least develop software for other organizations within the same business unit. In some cases a development organization cuts across multiple business units.

These multiple boundaries come with almost as many different types of relationships, which will affect exactly what types of transactions exist. In Table 1 I provide a list of some of the transaction types that I have identfied so far but it is by no means a complete list.

Table 1 – Some transaction types

Transaction type	**Specific types**	**Complexity**	**Transaction costs**
Organizational communication	Business case	Complexity increases with broader scope and additional stakeholders	Function of the management distance among the teams in the product line
	Concept of operations	Complexity increases with additional organizational boundaries within the scope	The CONOPS is likely to be more formal as the organizational relationships become more complex.
	Supplier management	Complexity increases with the complexity of the relationships	Preparation of statements of work or product specifications and acceptance test
	Partner management	Complexity increases with the complexity of the relationships	Negotiation of costs; preparation of contracts; cost of additional layer of separation
Release of software components	Binary distribution	Varying degrees of complexity among components; more complex as more variability supported	Documentation cost Help desk cost
	Source code distribution	Easy to become very complex	Cost of documenting associated deliverables such as build scripts

Release of non-software core assets (add additional assets that are important to the production system.)	Software architecture documentation; to a standard such as 1471 or less formal	Decreases as the documentation is more standardized.	Increases from informal to standard forms Help desk cost
	Formal model of interfaces	Directly related to complexity of the product	After initial education on notation, less cost than informal models
	Production plan	Varies with the number and type of variation mechanisms	Decreases as the production process becomes more automated.
Release of product	Executable plus supporting material including license texts, manuals	Less than that of the entire product line	Same as standard product
Feedback	Informal conversation w/partner	Very simple	Increases with level of response required
	Bug report	Routine	Increases with level of response required
	Feature request	Depends on the analysis	Increases with level of response required
Legal agreement	Accepting the license terms for a piece of Externally Available Software	The terms of the agreement determine the complexity; viral licenses that spread to other products introduce complexity	Actual dollars in some cases but at least the cost of including specific information in each product release
	Subcontractor relationship	Depends on whether fixed price or time and materials; previous relationship	Directly related to the complexity of the required deliverables

Each transaction has a cost that must be estimated by the modeler. The costs of a particular concept of operations can be computed once those transaction costs are determined. Various hypotheses can be investigated by reasoning about how the transaction costs will change with different assignment of tasks to groups on different sides of physical and legal boundaries.

4 BRIEF ANALYSIS

Lets do a brief analysis of the ecosystem for the Pedagogical Product Line (PPL) [SEI 09], developed by me for the SEI. I will omit parts to keep it a manageable size.

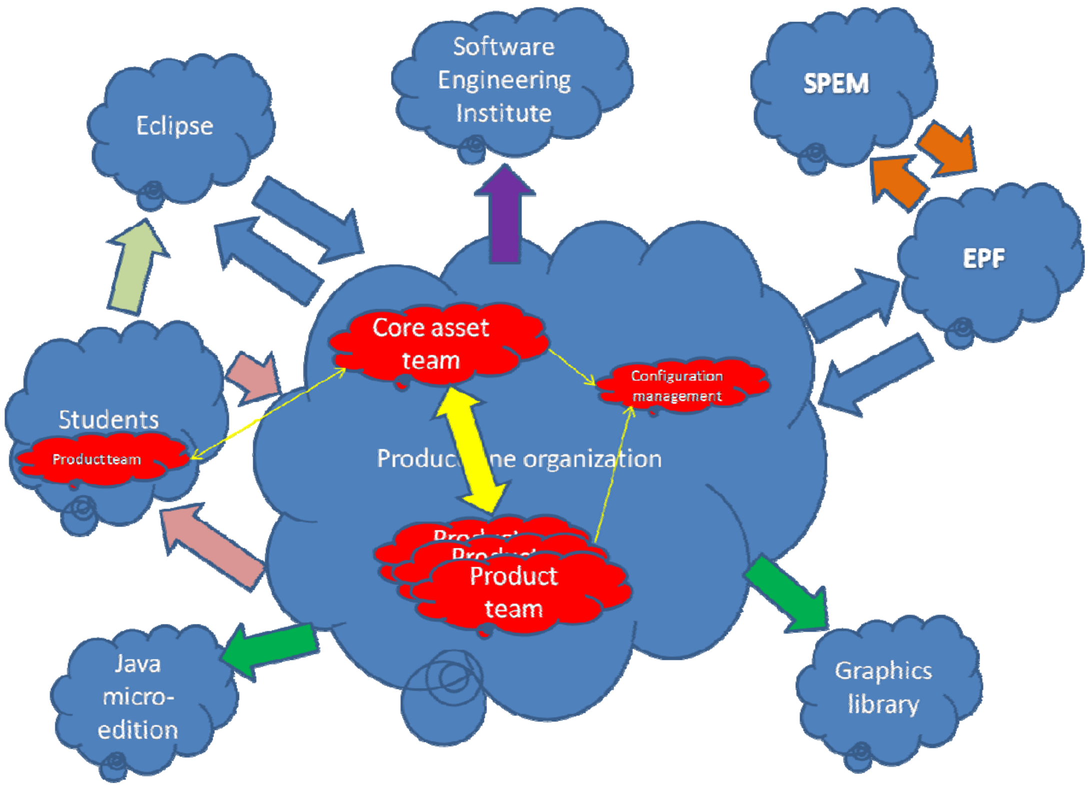

Figure 1 - Product line ecosystem

The pedagogical product line organization is the central player in the ecosystem depicted in Figure 1. The transactions and transfers include:

- Within the product line organization there is a transaction-free zone among the core asset team and product teams. There is also a transfer (no-cost) between the development teams and the configuration management system, which is automated. The interaction between the core asset team and product development teams was simple and no cost because there was only a minimal boundary between the groups. The core asset team provided a production plan to the product teams that reduced the costs of building a product.
- There is a transaction between the product line organization and the SEI. The costs for this transaction were significant. They included the time required for the SEI editors to give feedback and the time spent by the team to respond to those changes. This process took almost as long as the time to develop the assets originally.
- Eclipse [Eclipse 09] and the Eclipse Process Framework (EPF) [Eclipse 09b] are tools that were used by the product line organization. The product line team depended upon these tools and posted feature requests to those projects, one arrow. The costs to the team is the time required to learn the tools and the time required to establish the original automation, the other arrow.

- The students who are learning from the PPL use Eclipse to assemble core assets into products. The Eclipse ecosystem has produced many resources to help students learn to use Eclipse and this makes a class much easier to teach. The transaction of the students using the core assets is expensive. There is the cost of developing the production plan and the attached processes in the components. Since the PPL was deployed, a number of people have emailed with questions. These questions must be answered by the development team but the questions serve as the impetus to make changes to the materials.
- Finally, the Java micro-edition [Sun 09] and graphics libraries were used and the associated cost of that transaction was learning their capabilities. The Java micro-edition can be used in Eclipse by loading the correct jar files and setting Java Development Tools parameters in the IDE. In some cases, using externally available software such as this may impose licensing constraints.

Consider the ecosystem implications of one piece in this example. I used the EPF to capture the product line processes used to create the PPL. Readers who examine the PPl see references to the EPF, see a practical application of it and they in turn download the EPF and begin to use it. The PPL team makes feature requests to the EPF team and these lead to a better product (EPF) which attracts more users. EPF is an implementation of the Software Process Engineering Meta-model (SPEM) [OMG 08]. The EPF team participates in defining the standard and are guided in part by requests from users of EPF. New versions of SPEM drive new versions of the EPF. As more people adopt the standard they will use the tool.

5 TO BE CONTINUED

Most large software development organizations form some type of ecosystem that reaches beyond the boundaries of their unit in the organization and even beyond the limits of their corporate organization. Managers need to understand the architecture of that ecosystem to have some idea of the impact their strategic decisions will have on the ecosystem and the impact the evolution of the ecosytem will have on their unit. Using transaction theory we are able to help the manager develop that understanding by capturing the transactions among entities in the ecosystem.

A software product line organization has many of the characteristics of an ecosystem within itself but it also participates in other larger ecosystems. In the next issue of Strategic Software Engineering I will apply the ideas presented here to a software product line organization and will analyze several variations. I will also consider the implications for each of the practice areas in the SEI's Framework for Product Line Practice [SEI 09b].

6 ACKNOWLEDGEMENT

Thanks to Dr. Lawrence G. Jones of the Software Engineering Institute for valuable comments that allowed me to greatly improve this column.

REFERENCES

[Baldwin 07] Carliss Y. Baldwin. Modularity, Transactions, and the Boundaries of Firms: A Synthesis. Harvard Research Report 08-013, 2007.

[Brinkkemper 07] S. Brinkkemper, I. van Soest, and S. Jansen. Modeling of product software businesses: Investigation into industry product and channel typologies. In *proceedings of the Sixteenth International Conference on Information Systems Development*, pages 677–686. Springer-Verlag, 2007.

[Eclipse 09] Eclipse Foundation, www.eclipse.org.

[Eclipse 09b] Eclipse Foundation, Eclipse Process Framework, http://www.eclipse.org/epf, 2009.

[Messerschmitt 03] David G. Messerschmitt, Clemens Szyperski, Charles D. Ameringer. Software Ecosystem: Understanding an Indispensable Technology and Industry, MIT Press 2003.

[Microsoft 09] Welcome to the Windows Ecosystem Readiness Program, http://www.microsoft.com/whdc/Win7/default.mspx, 2009.

[Miller 06] Jeremey D. Miller. **Creating a Maintainable Software Ecosystem,** http://codebetter.com/blogs/jeremy.miller/archive/2006/08/13/148258.aspx.

[OMG 08] Object Management Group, Software & Systems Process Engineering Metamodel specification (SPEM), Version 2.0, http://www.omg.org/spec/SPEM/2.0/ 2008.

[SEI 09] Software Engineering Institute, Pedagogical Product Line (PPL), www.sei.cmu.edu/productlines/ppl, 2009.

[SEI 09b] Software Engineering Institute, A Framework for Software Product Line Practice, Version 5.0, http://www.sei.cmu.edu/productlines/framework.html, 2009.

[Sun 09] Sun, Java Micro Edition, http://java.sun.com/javame/index.jsp, 2009.

[Topcased 09] Topcased, www.topcased.org.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University, a visiting scientist at the Software Engineering Institute, and a partner in Luminary Software, a software engineering consulting firm. His research interests include software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.